

Rapport de tp02

Organisation :

La première réunion était pendant la séance de tp 02, suite à nous recherche du tp 01 on a opté pour le convertisseur pdf2txt qui était le plus adapter et pour la suite de notre travail on a commencé par choisir les langages maîtrisés par les membres de l'équipe et c'était Python , C, Ruby .

Ensuite, on a comparé le temps d'exécution de chacun de ces trois langage.

Pour la suite de notre travail la plupart de nous échanges était via un groupe discord et des petites réunions au Ciri dans notre temps libre afin de régler les problèmes qu'on a rencontré.

Comparaison de temps d'exécution :

Pour comparer le temps d'exécution de chaque langage on a exécuter le même code et on a calculer le temps d'exécution .

EN langage C :

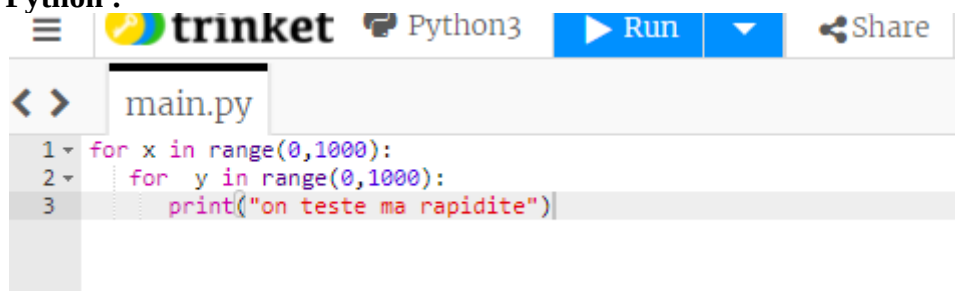
```
#include<stdio.h>

int main (){

    for (int i=0; i<1000; i++){
        for (int j=0; j<1000; j++){
            printf("on teste ma rapiditer \n");
        }
    }
}
```

Le temps d'exécution était de 108 secondes .

En langage Python :



Le temps d'exécution était de 30 secondes .

En langage Ruby :

```
1 for n in 0..1000
2   for m in 0..1000
3     print "on teste ma rapidite\n "
4   end
5 end
```

Le temps d'exécution était de 45 secondes .

Choix du langage :

On a remarqué qu'entre Python et Ruby il n'y avait pas beaucoup de différence en temps d'exécution ; après le vote de l'équipe on a choisi Ruby car c'était un nouveau langage pour nous qu'on voulait découvrir.

Le code en Ruby :

```
#!/usr/bin/env ruby
require 'pdf/reader'

# Starting time execution
start = Process.clock_gettime(Process::CLOCK_MONOTONIC)

ARGV.each do |filename|
  # Start converting using PDF-Reader
  PDF::Reader.open(filename) do |reader|

    puts "Converting : #{filename}"
    pageno = 0
    txt = reader.pages.map do |page|

      pageno += 1
      begin
        print "Converting Page #{pageno}/#{reader.page_count}\r"
        page.text
      rescue
        puts "Page #{pageno}/#{reader.page_count} Failed to convert"
        ..
      end
    end

  end

  puts "\nWriting text to disk"
```

```

    # Splitting the text to find the Abstract
    m = txt.join(",").split("\n\n")
    m.each do |i|
      i.strip!
    end

    # Finding the Abstract if it exists its title
    i = 0
    abstractIndex = 0
    introductionIndex = 0
    while i < m.length()
      if m[i].include?("Abstract") || m[i].include?("ABSTRACT")
        abstractIndex = i
      end
      i = i + 1
    end

    # Finding the next portion of the text that might be Introduction!
    i = 0
    while i < m.length()
      if m[i].include?("Introduction") || m[i].include?("introduction") || m[i].include?("INTRODUCTION")
        || m[i].to_s.include?("1.") || m[i].to_s.include?("I")
          introductionIndex = i
        end
        i = i + 1
      end
    end
```

```

# Merging the distance between the Abstract title and the Introduction title to have the portion of Abstract
abstract = ""
i = abstractIndex
if abstractIndex == 0
    abstract = "There is no Abstract to select!"
elseif abstractIndex && introductionIndex
    while i <= introductionIndex
        abstract += m[i].to_s
        i += 1
    end
end

# Inserting to the Text file
File.write filename+'.txt', "Filename: " + filename + "\nTitle: " + m.first.to_s + "\nAbstract: " + abstract
end # reader

end # each

# code to time
finish = Process.clock_gettime(Process::CLOCK_MONOTONIC)

diff = finish - start # Gets time in seconds as a float

puts "Execution Time is: #{diff} seconds!"

```

Exemple de résultat :

voici un exemple du résultat obtenu à l'exécution de ce code

```

ORPUS_TRAIN > ⚡ Das_Martins.pdf.txt
1  Filename: ./CORPUS_TRAIN/Das_Martins.pdf
2  Title: A Survey on Automatic Text Summarization
3  Abstract: Ono, K., Sumita, K., and Miike, S. (1994). Abstract generation based on rhetorical
4  structure extraction. In Proceedings of Coling '94, pages 344{348, Morristown,NJ, USA. [9]Osborne, M. (2002). Using maximum ent
5  of the ACL'02 Workshop on Automatic Summarization, pages 1{8, Morristown,
6  NJ, USA. [7]Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2001). Bleu: a method for
7  automatic evaluation of machine translation. In Proceedings of ACL '02, pages
8  311{318, Morristown, NJ, USA. [24]Radev, D. R., Hovy, E., and McKeown, K. (2002). Introduction to the special issue
9  on summarization. Computational Linguistics., 28(4):399{408. [1, 2]Radev, D. R., Jing, H., and Budzikowska, M. (2000). Centroid
10 studies. In NAACL-ANLP 2000 Workshop on Automatic summarization, pages
11 21{30, Morristown, NJ, USA. [12, 16, 17]30,Radev, D. R., Jing, H., Stys, M., and Tam, D. (2004). Centroid-based summariza-tion
12 40:919{938. [16, 17]Radev, D. R. and McKeown, K. (1998). Generating natural language summaries from multiple on-line sources. Co
13 automatic information. In Proceedings of SIGIR '88, pages 147{160, New York,NY, USA. [15]Salton, G., Wong, A., and Yang, A. C.
14 indexing. Communications of the ACM, 18:229{237. [20]Selman, B., Levesque, H. J., and Mitchell, D. G. (1992). A new method for
15 summarization by combining RankNet and third-party sources. In Proceedings of the EMNLP-CoNLL, pages 448{457. [7, 8]Witbrock, M.
16 a statistical approach to generating highly condensed non-extractive summaries. In Proceedings of SIGIR '99, pages 315{316, New

```

Explication du code :

Premièrement le fichier PDF a été converti à une data sous forme texte.

Ensuite pour récupérer le titre, le nom et le résumé (abstract) du fichier on a divisé cette data à une table pour chercher le mots clé abstract. Et on a fait des boucles pour les mettre en ordre. à la fin, toutes les informations qui existent, ils vont être envoyés dans un fichier texte.

En plus le temps d'exécution avant et après de ce processus a été calculé.

Le temps d'exécution :

Pour chaque fichier, la moyenne du temps d'exécution était environ 2.5 seconds.

Les problèmes :

On a eu des problèmes durant la conversion des articles de deux colonnes. Par exemple on a pas pu récupérer l'abstract car il a pas bien effectué la recherche.

L'Avantage :

C'était la première fois qu'on travaille avec Ruby, la syntaxe est très simple, proche de Python et il est très rapide en exécution.